UDACITY

< Return to Classroom

DISCUSS ON STUDENT HUB

Facial Keypoint Detection

REVIEW

CODE REVIEW

HISTORY

Requires Changes

3 specifications require changes

Files Submitted

The submission includes models.py and the following Jupyter notebooks, where all questions have been answered and training and visualization cells have been executed:
2. Define the Network Architecture.ipynb, and
3. Facial Keypoint Detection, Complete Pipeline.ipynb.
Other files may be included, but are not necessary for grading purposes. Note that all your files will be zipped and uploaded should you submit via the provided workspace.

Great job submitting all the required files

`models.py`

Define a convolutional neural network with at least one convolutional layer, i.e. `self.conv1 = nn.Conv2d(1, 32, 5)`. The network should take in a grayscale, square image.

Brilliant job adding dropout to avoid overfitting. You can also checkout batch normalization which also reduces overfitting.
Another way to improve your network is by using Transfer Learning. If you are interested, you can check out this tutorial

Notebook 2: Define the Network Architecture

Define a `data_transform` and apply it whenever you instantiate a DataLoader. The composed transform should include: rescaling/cropping, normalization, and turning input images into torch Tensors. The transform should turn any input image into a normalized, square, grayscale image and then a Tensor for your model to take it as input.

Depending on the complexity of the network you define, and other hyperparameters the model can take some time to train. We encourage you to start with a simple network with only 2 layers. You'll be graded based on the implementation of your models rather than accuracy.

Nice work rescaling, cropping, normalizing and converting your image to a tensor. You might want to consider augmenting the training data by randomly rotating and/or flipping the images. You can read more about the different transforms you can apply from the official documentation

Select a loss function and optimizer for training the model. The loss and optimization functions should be appropriate for keypoint detection, which is a regression problem.

Awesome job choosing the L1Loss as the loss function and SGD optimizer.
L1Loss is a great choice because this is a regression problem and it compares the output value by value and not by looking at the probability distribution which is the case with classification problems.

Train your CNN after defining its loss and optimization functions. You are encouraged, but not required, to visualize the loss over time/epochs by printing it out occasionally and/or plotting the loss over time. Save your best trained model.

Great attempt training your network. However, the results are not acceptable as the facial keypoints seem scattered and do not represent the structure of a face. Please do the following to improve the performance of your network.

- Train your network for longer say atleast 15 epochs

- Train your network for longer say atleast 15 epochs
- Increase the depth of your network. You could start with the Naimish network that is suggested in Notebook 2.
- Reduce your learning rate and training again.
- Use the Adam as the optimizer and SmoothL1Loss as the loss as they have proven to show great results.

After training, all 3 questions about model architecture, choice of loss function, and choice of batch_size and epoch parameters are answered.

Your CNN "learns" (updates the weights in its convolutional layers) to recognize features and this criteria requires that you extract at least one convolutional filter from your trained model, apply it to an image, and see what effect this filter has on an image.

The CNN does "learn" to recognize the features in the image

After visualizing a feature map, answer: what do you think it detects? This answer should be informed by how a filtered image (from the criteria above) looks.

Great reasoning and intuition on what the filter detects. 👌
In addition to detecting vertical edges, it also blurs the image.

Notebook 3: Facial Keypoint Detection

Use a Haar cascade face detector to detect faces in a given image.

Awesome job using the Haar cascade face detector to detect faces in the image

You should transform any face into a normalized, square, grayscale image and then a Tensor for your model to take in as input (similar to what the `data_transform` did in Notebook 2).

Great attempt here. Your error probably begins from the line:

```
    roi = image_with_detections[y:y+h, x:x+w]
```

Rather, you should create a copy of the `image` and the work with that image copy. Something like:

```
image_copy = np.copy(image)
# loop over the detected faces from your haar cascade
for (x,y,w,h) in faces:
    # Select the region of interest that is the face in the image
    roi = image_copy[y-10:y+h+10, x-8:x+w+8]
```

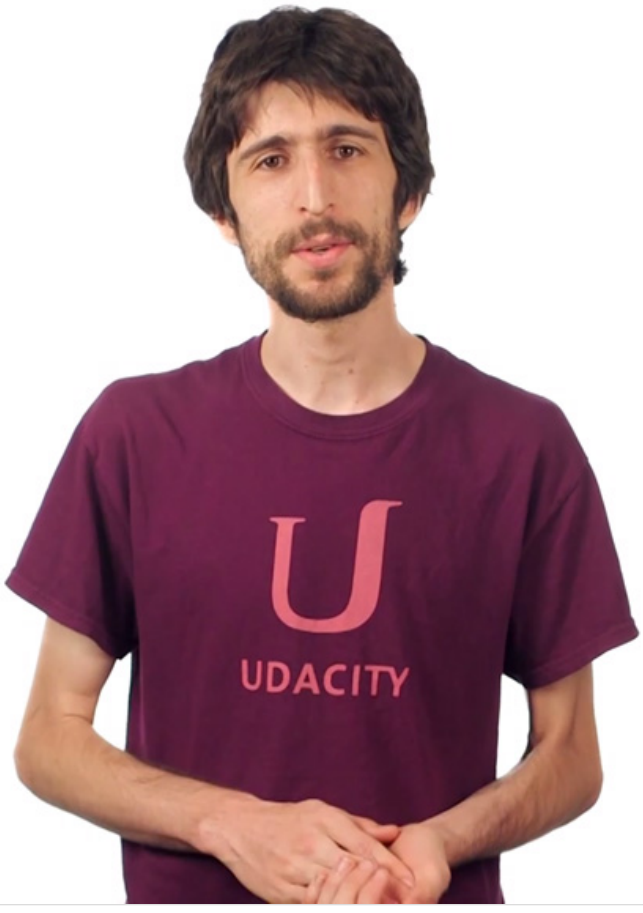
If you encounter any errors after this, please ask questions on Knowledge and a mentor will help you out.

After face detection with a Haar cascade and face pre-processing, apply your trained model to each detected face, and display the predicted keypoints for each face in the image.

Please implement the corrections pointed out above.

RESUBMIT

DOWNLOAD PROJECT



Best practices for your project resubmission

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH

Rate this review

START